Data everywhere
Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)
Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what??

Monday, August 23, 2010
Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what??

Monday, August 23, 2010
Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what??

Monday, August 23, 2010
Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

more is: different!

So what ??
Data everywhere

- Opportunities
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - “Brute-force” methods may suffice

- Challenges
  - “Dirtier” data
  - Efficient algorithms
  - Scalability (with reasonable cost)
Data everywhere

- **Opportunities**
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - “Brute-force” methods may suffice

- **Challenges**
  - “Dirtier” data
  - Efficient algorithms
  - Scalability (with reasonable cost)
Data everywhere

- Opportunities
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - “Brute-force” methods may suffice

- Challenges
  - “Dirtier” data
  - Efficient algorithms
  - Scalability (with reasonable cost)
“The Google Way”

“All models are wrong, but some are useful”
– George Box
“The Google Way”

“All models are wrong, but some are useful”
– George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
“The Google Way”

“All models are wrong, but some are useful”
– George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
- Language translation
“The Google Way”

“All models are wrong, but some are useful” – George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
- Language translation
- …

Chris Anderson, Wired (July 2008)
Getting over the marketing hype…

Cloud Computing =
Getting over the marketing hype…

Cloud Computing

= Internet
Getting over the marketing hype…

Cloud Computing = Internet + Commoditization/standardization
Getting over the marketing hype…

Cloud Computing = Internet + Commoditization/standardization

‘It’s what I and many others have worked towards our entire careers. It’s just happening now.’

— Eric Schmidt
This tutorial

- Is not about cloud computing
- But about large scale data processing
This tutorial

- **Is not** about cloud computing
- **But** about large scale data processing

Data + Algorithms
Tutorial overview

- Part 1 (Spiros): Basic concepts & tools
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive

- Part 2 (Jimeng): Algorithms
  - Information retrieval
  - Graph algorithms
  - Clustering (k-means)
  - Classification (k-NN, naïve Bayes)

- Part 3 (Rong): Applications
  - Text processing
  - Data warehousing
  - Machine learning

Monday, August 23, 2010
Outline

- Introduction
- **MapReduce & distributed storage**
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
What is MapReduce?

- Programming model?
- Execution environment?
- Software package?
What is MapReduce?

- Programming model?
- Execution environment?
- Software package?

It’s all of those things, depending who you ask...
What is MapReduce?

- Programming model?
- Execution environment?
- Software package?

“MapReduce” (this talk) == Distributed computation + distributed storage + scheduling / fault tolerance

It’s all of those things, depending who you ask...
Example – Programming model

employees.txt

<table>
<thead>
<tr>
<th># Last</th>
<th>First</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

Q: “What is the frequency of each first name?”
Example – Programming model

employees.txt

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

... ... ...

Q: “What is the frequency of each first name?”

def getName (line):

### Example – Programming model

**employees.txt**

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

... ... ... ...

Q: "What is the frequency of each first name?"

```python
def getName(line):
    return line.split(' \t')[1]

def addCounts(hist, name):
```
Example – Programming model

Q: “What is the frequency of each first name?”

```python
# employees.txt

# LAST   FIRST   SALARY
Smith    John    $90,000
Brown    David   $70,000
Johnson  George  $95,000
Yates    John    $80,000
Miller   Bill    $65,000
Moore    Jack    $85,000
Taylor   Fred    $75,000
Smith    David   $80,000
Harris   John    $90,000
...
...
...
...

mapper
def getName(line):
    return line.split(‘\t’)[1]

def addCounts(hist, name):
    hist[name] =
        hist.get(name, default=0) + 1
    return hist

input = open(‘employees.txt’, ‘r’)```
Example – Programming model

Q: "What is the frequency of each first name?"

```python
employees.txt

<table>
<thead>
<tr>
<th># LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

mapper

def getName(line):
    return line.split('\t')[1]

def addCounts(hist, name):
    hist[name] = \
    hist.get(name, default=0) + 1
    return hist

input = open('employees.txt', 'r')
```
employees.txt

<table>
<thead>
<tr>
<th># LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

mapper

```python
def getName(line):
    return line.split(\'\t\')[1]
```

def addCounts(hist, name):
    hist[name] = \
    hist.get(name, default=0) + 1
    return hist

input = open(\'employees.txt\', \'r\')

intermediate = map(getName, input)

Q: “What is the frequency of each first name?”
Example – Programming model

employees.txt

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

mapper

def getName (line):
    return line.split('t')[1]

def addCounts (hist, name):
    hist[name] =
    hist.get(name,default=0) + 1
    return hist

input = open('employees.txt', 'r')
intermediate = map(getName, input)

Q: “What is the frequency of each first name?”
Example – Programming model

employees.txt

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

mapper

```python
def getName(line):
    return line.split(\'\t\')[1]
```

reducer

```python
def addCounts(hist, name):
    hist[name] = \
    hist.get(name, default=0) + 1
    return hist
```

```python
input = open(\'employees.txt\', \'r\')

intermediate = map(getName, input)

result = reduce(addCounts, \nintermediate, {})
```
Example – Programming model

```python
def getName (line):
    return (line.split('t') [1], 1)

def addCounts(hist, (name, c)):
    hist[name] = \n    hist.get(name,default=0) + c
    return hist

input = open('employees.txt', 'r')
intermediate = map(getName, input)
result = reduce(addCounts, \n    intermediate, {})
```

**employees.txt**

<table>
<thead>
<tr>
<th># LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

Q: “What is the frequency of each first name?”
Example – Programming model

```python
def getName(line):
    return (line.split('t')[1], 1)
def addCounts(hist, (name, c)):
    hist[name] = hist.get(name, default=0) + c
    return hist

input = open('employees.txt', 'r')
intermediate = map(getName, input)
result = reduce(addCounts, 

Q: “What is the frequency of each first name?”
```
public class HistogramJob extends Configured implements Tool {

public static class FieldMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private static LongWritable ONE = new LongWritable(1);
    private static Text firstname = new Text();

    @Override
    public void map (LongWritable key, Text value,
                     OutputCollector<Text, LongWritable> out, Reporter r) {
        firstname.set(value.toString().split("\t")[1]);
        output.collect(firstname, ONE);
    }

} // class FieldMapper
public class HistogramJob extends Configured implements Tool {

    public static class FieldMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, LongWritable> {

        private static LongWritable ONE = new LongWritable(1);
        private static Text firstname = new Text();

        @Override
        public void map (LongWritable key, Text value, OutputCollector<Text, LongWritable> out, Reporter r) {
            firstname.set(value.toString().split("\t")[1]);
            output.collect(firstname, ONE);
        }
    } // class FieldMapper

    // Example – Programming model
    Hadoop / Java

    // non-boilerplate

    Monday, August 23, 2010
public class HistogramJob extends Configured implements Tool {

public static class FieldMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, LongWritable> {

private static LongWritable ONE = new LongWritable(1);
private static Text firstname = new Text();

@Override
public void map (LongWritable key, Text value, OutputCollector<Text, LongWritable> out, Reporter r) {
firstname.set(value.toString().split("\t")[1]);
output.collect(firstname, ONE);
}
} // class FieldMapper

Example – Programming model
Hadoop / Java
Example – Programming model
Hadoop / Java

```java
public static class LongSumReducer extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private static LongWritable sum = new LongWritable();

    @Override
    public void reduce(Text key, Iterator<LongWritable> vals,
                        OutputCollector<Text, LongWritable> out, Reporter r) {
        long s = 0;
        while (vals.hasNext())
            s += vals.next().get();
        sum.set(s);
        output.collect(key, sum);
    }
} // class LongSumReducer
```
Example – Programming model
Hadoop / Java

```java
public static class LongSumReducer extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private static LongWritable sum = new LongWritable();

    @Override
    public void reduce (Text key, Iterator<LongWritable> vals,
                        OutputCollector<Text, LongWritable> out, Reporter r) {
        long s = 0;
        while (vals.hasNext())
            s += vals.next().get();
        sum.set(s);
        output.collect(key, sum);
    }

} // class LongSumReducer
```
Example – Programming model
Hadoop / Java

```java
public int run (String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), HistogramJob.class);
    job.setJobName("Histogram");
    FileInputFormat.setInputPaths(job, args[0]);
    job.setMapperClass(FieldMapper.class);
    job.setCombinerClass(LongSumReducer.class);
    job.setReducerClass(LongSumReducer.class);
    // ...
    JobClient.runJob(job);
    return 0;
} // run()

public static main (String[] args) throws Exception {
    ToolRunner.run(new Configuration(), new HistogramJob(), args)
} // main()
} // class HistogramJob
```
Example – Programming model
Hadoop / Java

```java
public int run (String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), HistogramJob.class);
    job.setJobName("Histogram");
    FileInputFormat.setInputPaths(job, args[0]);
    job.setMapperClass(FieldMapper.class);
    job.setCombinerClass(LongSumReducer.class);
    job.setReducerClass(LongSumReducer.class);
    // ...
    JobClient.runJob(job);
    return 0;
} // run()

public static main (String[] args) throws Exception {
    ToolRunner.run(new Configuration(), new HistogramJob(), args);
} // main()
} // class HistogramJob
```

~ 30 lines = 25 boilerplate (Eclipse) + 5 actual code
MapReduce for...

- Distributed clusters
  - Google’s original
  - Hadoop (Apache Software Foundation)

- Hardware
  - SMP/CMP: Phoenix (Stanford)
  - Cell BE

- Other
  - Skynet (in Ruby/DRB)
  - QtConcurrent
  - BashReduce
  - …many more
MapReduce for...

Distributed clusters
- Google’s original
- Hadoop (Apache Software Foundation)

Hardware
- SMP/CMP: Phoenix (Stanford)
- Cell BE

Other
- Skynet (in Ruby/DRB)
- QtConcurrent
- BashReduce
- ...many more
Recap

<table>
<thead>
<tr>
<th>Quick-n-dirty script</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>~5 lines of (non-boilerplate) code</td>
<td>Up to <em>thousands</em> of machines and drives</td>
</tr>
<tr>
<td>Single machine, local drive</td>
<td></td>
</tr>
</tbody>
</table>
Recap

<table>
<thead>
<tr>
<th>Quick-n-dirty script</th>
<th>vs</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>~5 lines of (non-boilerplate) code</td>
<td></td>
<td>Up to <em>thousands</em> of machines and drives</td>
</tr>
<tr>
<td>Single machine, local drive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is hidden to achieve this:

- Data partitioning, placement and replication
- Computation placement (and replication)
- Number of nodes (mappers / reducers)
- ...

Monday, August 23, 2010
Recap

<table>
<thead>
<tr>
<th>Quick-n-dirty script</th>
<th>vs</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>~5 lines of (non-boilerplate) code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single machine, local drive</td>
<td></td>
<td>Up to <em>thousands</em> of machines and drives</td>
</tr>
</tbody>
</table>

What is hidden to achieve this:

- Data partitioning, placement and replication
- Computation placement (and replication)
- Number of nodes (mappers / reducers)

As a programmer, you don’t *need* to know what I’m about to show you next...

Monday, August 23, 2010
Execution model: Flow

Input file

- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Mapper

Mapper

Mapper

Reducer

Reducer

Mapper

Output file

- PART 0
- PART 1
Execution model: Flow
Execution model: Flow

Input file:
- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Sequential scan

Key/value iterators

Mapper
- MAPPER
- MAPPER
- MAPPER

Reducer
- REDUCER
- REDUCER

Output file:
- PART 0
- PART 1

Monday, August 23, 2010
Execution model: Flow

Input file

SPLIT 0

SPLIT 1

SPLIT 2

SPLIT 3

Sequential scan

Key/value iterators

Mapper

Reducer

Mapper

Mapper

Mapper

Reducer

Mapper

Output file

PART 0

PART 1

Input file entries:
- Smith John $90,000
- Yates John $80,000
Execution model: Flow

Input file

SPLIT 0
SPLIT 1
SPLIT 2
SPLIT 3

Key/value iterators

MAPPER
MAPPER
MAPPER

REDUCER
REDUCER

Output file

PART 0
PART 1

Sequential scan

Monday, August 23, 2010
Execution model: Flow

Input file
- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Key/value iterators

Mapper
- John 1

Reducer

Output file
- PART 0
- PART 1

Sequential scan

All-to-all, hash partitioning

Monday, August 23, 2010
Execution model: Flow

Input file

- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Mapper

- John 2

Reducer

Key/value iterators

Output file

- PART 0
- PART 1

Sequential scan

Sort-merge

All-to-all, hash partitioning

Monday, August 23, 2010
Execution model: Flow

Input file
- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Mapper
- SPLIT 0
- SPLIT 1
- SPLIT 2
- SPLIT 3

Reducer
- PART 0
- PART 1

Key/value iterators

Sequential scan

Sort-merge

All-to-all, hash partitioning

Output file
- John
- 2

Monday, August 23, 2010
Execution model: Placement

HOST 0
- SPLIT 0: Replica 1/3
- SPLIT 1: Replica 2/3
- SPLIT 3: Replica 2/3

HOST 1
- SPLIT 0: Replica 2/3
- SPLIT 4: Replica 1/3

HOST 2
- SPLIT 3: Replica 3/3
- SPLIT 2: Replica 2/3

HOST 3
- SPLIT 2: Replica 3/3
- SPLIT 1: Replica 1/3

HOST 4

HOST 5

HOST 6

Monday, August 23, 2010
Execution model: Placement

HOST 0

SPLIT 0 Replica 1/3
SPLIT 1 Replica 2/3
SPLIT 3 Replica 2/3

MAPPER

HOST 1

SPLIT 0 Replica 2/3
SPLIT 4 Replica 1/3
SPLIT 3 Replica 1/3

MAPPER

HOST 2

SPLIT 3 Replica 3/3
SPLIT 2 Replica 2/3
SPLIT 0 Replica 3/3

MAPPER

HOST 3

SPLIT 2 Replica 3/3
SPLIT 1 Replica 1/3
SPLIT 4 Replica 2/3

HOST 4

HOST 5

HOST 6

Computation co-located with data (as much as possible)

Monday, August 23, 2010
Execution model: Placement

HOST 0
- SPLIT 0: Replica 1/3
- SPLIT 1: Replica 2/3
- SPLIT 3: Replica 2/3

HOST 1
- SPLIT 0: Replica 2/3
- SPLIT 4: Replica 1/3

HOST 2
- SPLIT 3: Replica 3/3
- SPLIT 2: Replica 2/3

HOST 3
- SPLIT 2: Replica 3/3
- SPLIT 1: Replica 1/3
- SPLIT 4: Replica 2/3

HOST 4

HOST 5

HOST 6

MAPPER
Execution model: Placement

HOST 0
- SPLIT 0 Replica 1/3
- SPLIT 1 Replica 2/3
- SPLIT 3 Replica 2/3
- MAFPER

HOST 1
- SPLIT 0 Replica 2/3
- SPLIT 4 Replica 1/3
- MAFPER
- SPLIT 3 Replica 1/3
- REDUCER

HOST 2
- SPLIT 3 Replica 3/3
- SPLIT 2 Replica 2/3
- MAFPER
- SPLIT 0 Replica 3/3

HOST 3
- SPLIT 2 Replica 3/3
- SPLIT 1 Replica 1/3
- MAFPER
- SPLIT 4 Replica 2/3

HOST 4
- SPLIT 2 Replica 3/3

HOST 5
- SPLIT 2 Replica 3/3

HOST 6
- SPLIT 2 Replica 3/3
- REDUCER

Monday, August 23, 2010
Execution model: Placement

Rack/network-aware

Monday, August 23, 2010
MapReduce Summary
MapReduce Summary

- Simple programming model
- Scalable, fault-tolerant
- Ideal for (pre-)processing large volumes of data
MapReduce Summary

- Simple programming model
- Scalable, fault-tolerant
- Ideal for (pre-)processing large volumes of

‘However, if the data center is the computer, it leads to the even more intriguing question “What is the equivalent of the ADD instruction for a data center?” […] If MapReduce is the first instruction of the “data center computer”, I can’t wait to see the rest of the instruction set, as well as the data center programming language, the data center operating system, the data center storage systems, and more.’

Outline

- Introduction
- MapReduce & distributed storage
- **Hadoop**
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary

Monday, August 23, 2010
Hadoop

- HBase
- Pig
- Hive
- Chukwa

- MapReduce
- HDFS
- ZooKeeper

- Core
- Avro
Hadoop’s stated mission (Doug Cutting interview):
Commoditize infrastructure for web-scale, data-intensive applications
Who uses Hadoop?

- Yahoo!
- Facebook
- Last.fm
- Rackspace
- Digg

- Apache Nutch

- ... more in part 3
Hadoop

- HBase
- Pig
- Hive
- Chukwa
- MapReduce
- HDFS
- ZooKeeper
- Core
- Avro

Monday, August 23, 2010
Hadoop

Filesystems and I/O:
- Abstraction APIs
- RPC / Persistence

Core
Avro

HBase
Pig
Hive
Chukwa

ZooKeeper
Hadoop

- HBase
- Pig
- Hive
- Chukwa
- MapReduce
- HDFS
- Zoo Keeper
- Core
- Avro

Monday, August 23, 2010
Hadoop

Cross-language serialization:
- RPC / persistence
- \sim Google ProtoBuf / FB Thrift

Monday, August 23, 2010
Hadoop
Hadoop

Distributed execution (batch)
- Programming model
- Scalability / fault-tolerance

- MapReduce
- HDFS
- ZooKeeper
- Core
- Avro
Hadoop

- HBase
- Pig
- Hive
- Chukwa

- MapReduce
- HDFS
- ZooKeeper

- Core
- Avro
Hadoop

Hadoop

MapReduce

HDFS

ZooKeeper

Core

Avro

Distributed storage (read-opt.)
- Replication / scalability
- ~ Google filesystem (GFS)
Hadoop

HBase  Pig  Hive  Chukwa

MapReduce  HDFS  Zoo Keeper

Core  Avro
Hadoop

- HBase
- MapReduce
- HDFS
- Core
- Avro
- Coordination service
  - Locking / configuration
  - ~ Google Chubby
- ZooKeeper
Hadoop

- HBase
- Pig
- Hive
- Chukwa

- MapReduce
- HDFS
- Zoo Keeper

- Core
- Avro
Hadoop

- HBase
- Pig
- Hive
- Chukwa
- ZooKeeper

Column-oriented, sparse store
- Batch & random access
- ~ Google BigTable

Core

Avro
Hadoop

Data flow language
- Procedural SQL-inspired lang.
- Execution environment

Monday, August 23, 2010
Hadoop

- HBase
- Pig
- Hive
- Chukwa
- MapReduce
- HDFS
- ZooKeeper
- Core
- Avro
Hadoop

HBase   Pig    Hive   Chukwa

Distributed data warehouse
- SQL-like query language
- Data mgmt / query execution

Core   Avro
Hadoop

- HBase
- Pig
- Hive
- Chukwa
- MapReduce
- HDFS
- ZooKeeper
- Core
- Avro

... ... more
MapReduce

- **Mapper:** \((k_1, v_1) \rightarrow (k_2, v_2)[]\)
  - E.g., \((\text{void}, \text{textline} : \text{string}) \rightarrow (\text{first} : \text{string}, \text{count} : \text{int})\)

- **Reducer:** \((k_2, v_2[]) \rightarrow (k_3, v_3)[]\)
  - E.g., \((\text{first} : \text{string}, \text{counts} : \text{int[]} \rightarrow (\text{first} : \text{string}, \text{total} : \text{int})\)

- **Combiner:** \((k_2, v_2[]) \rightarrow (k_2, v_2)[]\)

- **Partition:** \((k_2, v_2) \rightarrow \text{int}\)
Mapper interface

```java
interface Mapper<K1, V1, K2, V2> {
    void configure (JobConf conf);
    void map (K1 key, V1 value,
              OutputCollector<K2, V2> out,
              Reporter reporter);
    void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key,val)` any time
Reducer interface

```
interface Reducer<K2, V2, K3, V3> {
    void configure (JobConf conf);
    void reduce (K2 key, Iterator<V2> values, OutputCollector<K3, V3> out, Reporter reporter);
    void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key, val)` any time
Some canonical examples

- Histogram-type jobs:
  - Graph construction (bucket = edge)
  - K-means et al. (bucket = cluster center)

- Inverted index:
  - Text indices
  - Matrix transpose

- Sorting
- Equi-join

- More details in part 2
Equi-joins
“Reduce-side”

(Smith, 7)
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

(Sales, 3)
(Devel, 7)
(Acct., 5)
Equi-joins

“Reduce-side”

(Smith, 7)
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

(Sales, 3)
(Devel, 7)
(Acct., 5)

7: (Smith)
Equi-joins
“Reduce-side”

(Smith, 7)  MAP  7: (■, (Smith))
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

(Sales, 3)  MAP  7: (■, (Devel))
(Devel, 7)
(Acct., 5)
Equi-joins
“Reduce-side”

(Smith, 7)
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

7: ([■], (Smith))
-OR-
(7,■): (Smith)

(Sales, 3)
(Devel, 7)
(Acct., 5)

7: ([■], (Devel))
-OR-
(7,■): (Devel)
## Equi-joins

“Reduce-side”

<table>
<thead>
<tr>
<th>(Smith, 7)</th>
<th>7: (◼, (Smith))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Jones, 7)</td>
<td>7: (◼, (Jones))</td>
</tr>
<tr>
<td>(Brown, 7)</td>
<td>7: (◼, (Brown))</td>
</tr>
<tr>
<td>(Davis, 3)</td>
<td></td>
</tr>
<tr>
<td>(Dukes, 5)</td>
<td></td>
</tr>
<tr>
<td>(Black, 3)</td>
<td></td>
</tr>
<tr>
<td>(Gruhl, 7)</td>
<td>7: (◼, (Gruhl))</td>
</tr>
<tr>
<td>(Sales, 3)</td>
<td>7: (◼, (Devel))</td>
</tr>
<tr>
<td>(Devel, 7)</td>
<td></td>
</tr>
<tr>
<td>(Acct., 5)</td>
<td></td>
</tr>
</tbody>
</table>
Equi-joins
“Reduce-side”

7: (Smith, 7)
7: (Jones, 7)
7: (Brown, 7)
7: (Davis, 3)
7: (Dukes, 5)
7: (Black, 3)
7: (Gruhl, 7)

7: (Smith, 7)
7: (Jones, 7)
7: (Brown, 7)

7: (Smith, 7)
7: (Jones, 7)
7: (Brown, 7)
7: (Gruhl, 7)
7: (Devel, 7)

7: { (Smith, 7), (Jones, 7), (Brown, 7), (Gruhl, 7), (Devel, 7) }
Equi-joins
“Reduce-side”

(Smith, 7)
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

(Sales, 3)
(Devel, 7)
(Acct., 5)

7: (◼, (Smith))
7: (◼, (Jones))
7: (◼, (Brown))

7: (◼, (Gruhl))

7: (◼, (Devel))

7: { (◼, (Smith)),
      (◼, (Jones)),
      (◼, (Brown)),
      (◼, (Gruhl)),
      (◼, (Devel)) }
HDFS & MapReduce processes
HDFS & MapReduce processes
HDFS & MapReduce processes
Hadoop Streaming & Pipes

- Don’t have to use Java for MapReduce

- Hadoop Streaming:
  - Use stdin/stdout & text format
  - Any language (C/C++, Perl, Python, shell, etc)

- Hadoop Pipes:
  - Use sockets & binary format (more efficient)
  - C++ library required
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
HBase introduction

- MapReduce canonical example:
  - Inverted index (more in Part 2)

- Batch computations on large datasets:
  - Build static index on crawl snapshot

- However, in reality crawled pages are:
  - Updated by crawler
  - Augmented by other parsers/analytics
  - Retrieved by cache search
  - Etc…
HBase introduction

- MapReduce & HDFS:
  - Distributed storage + computation
  - Good for batch processing
  - But: no facilities for accessing or updating individual items

- HBase:
  - Adds random-access read / write operations
  - Originally developed at Powerset
  - Based on Google’s Bigtable
HBase data model

Partitioned over many nodes
HBase data model

Column family (hundreds; fixed)

Row (billions; sorted)

Partitioned over many nodes (thousands)
### HBase data model

<table>
<thead>
<tr>
<th>Column family (hundreds; fixed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column (millions)</td>
</tr>
</tbody>
</table>

- Keys and cell values are arbitrary byte arrays
- Partitioned over many nodes (thousands)
- Row (billions; sorted)
- Key
HBase data model

Column (millions)

Column family (hundreds; fixed)

Keys and cell values are arbitrary byte arrays

Row (billions; sorted)

Can use any underlying data store (local, HDFS, S3, etc)

Partitioned over many nodes (thousands)
### Data model example

#### profile: family

<table>
<thead>
<tr>
<th>profile: last</th>
<th>profile: first</th>
<th>profile: salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>
# Data model example

<table>
<thead>
<tr>
<th>profile:last</th>
<th>profile:first</th>
<th>profile:salary</th>
<th>bm:url1</th>
<th>bm:urlN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Data model example**

<table>
<thead>
<tr>
<th>profile: last</th>
<th>profile: first</th>
<th>profile: salary</th>
<th>bm: url1</th>
<th>bm: urlN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Always access via primary key

Monday, August 23, 2010
HBase vs. RDBMS

- Different solution, similar problems
- RDBMSes:
  - Row-oriented
  - Fixed-schema
  - ACID
- HBase et al.:
  - Designed from ground-up to scale out, by adding commodity machines
  - Simple consistency scheme: atomic row writes
  - Fault tolerance
  - Batch processing
  - No (real) indexes
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Pig introduction

- “~5 lines of non-boilerplate code”
- Writing a single MapReduce job requires significant gruntwork
  - Boilerplates (mapper/reducer, create job, etc)
  - Input / output formats
- Many tasks require more than one MapReduce job
Pig main features

- Data structures (multi-valued, nested)
- Pig-latin: data flow language
  - SQL-inspired, but imperative (not declarative)
Pig example

records = LOAD filename AS (last: chararray, first: chararray, salary:int);
grouped = GROUP records BY first;
counts = FOREACH grouped GENERATE group, COUNT(records.first);
DUMP counts;

Q: “What is the frequency of each first name?”

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig schemas

- Schema = tuple data type

- Schemas are optional!
  - Data-loading step is not required
  - “Unknown” schema: similar to AWK ($0, $1, ..)

- Support for most common datatypes
- Support for nesting
Pig Latin feature summary

- Data loading / storing
  - LOAD / STORE / DUMP

- Filtering
  - FILTER / DISTINCT / FOREACH / STREAM

- Group-by
  - GROUP

- Join & co-group
  - JOIN / COGROUP / CROSS

- Sorting
  - ORDER / LIMIT

- Combining / splitting
  - UNION / SPLIT
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Cascading introduction

- Provides higher-level abstraction
  - Fields, Tuples
  - Pipes
  - Operations
  - Taps, Schemes, Flows

- Ease composition of multi-job flows
Cascading introduction

- Provides higher-level abstraction
  - Fields, Tuples
  - Pipes
  - Operations
  - Taps, Schemes, Flows
- Ease composition of multi-job flows

Library, not a new language
Cascading example

Scheme srcScheme = new TextLine();
Tap source = new Hfs(srcScheme, filename);
Scheme dstScheme = new TextLine();
Tap sink = new Hfs(dstScheme, filename, REPLACE);

Pipe assembly = new Pipe(“lastnames”);

Function splitter = new RegexSplitter(
    new Fields(“last”, “first”, “salary”), “\t”);
assembly = new Each(assembly, new Fields(“line”), splitter);

assembly = new GroupBy(assembly, new Fields(“first”));

Aggregator count = new Count(new Fields(“count”));
assembly = new Every(assembly, count);

FlowConnector flowConnector = new FlowConnector();
Flow flow = flowConnector.connect(“last-names”,
    source, sink, assembly);
flow.complete();

# LAST FIRST SALARY
Smith John $90,000
Brown David $70,000
... ... ...
employees.txt
Q: “What is the frequency of each first name?”
Cascading example

Scheme srcScheme = new TextLine();
Tap source = new Hfs(srcScheme, filename);
Scheme dstScheme = new TextLine();
Tap sink = new Hfs(dstScheme, filename, REPLACE);

Pipe assembly = new Pipe(“lastnames”);

Function splitter = new RegexSplitter(
    new Fields(“last”, “first”, “salary”), “\t”);
assembly = new Each(assembly, new Fields(“line”), splitter);
assembly = new GroupBy(assembly, new Fields(“first”));

Aggregator count = new Count(new Fields(“count”));
assembly = new Every(assembly, count);

FlowConnector flowConnector = new FlowConnector();
Flow flow = flowConnector.connect(“last-names”,
    source, sink, assembly);
flow.complete();

Q: “What is the frequency of each first name?”
Cascading feature summary

- Pipes: transform streams of tuples
  - Each
  - GroupBy / CoGroup
  - Every
  - SubAssembly

- Operations: what is done to tuples
  - Function
  - Filter
  - Aggregator / Buffer
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Hive introduction

- Originally developed at Facebook
  - Now a Hadoop sub-project
- Data warehouse infrastructure
  - Execution: MapReduce
  - Storage: HDFS files
- Large datasets, e.g. Facebook daily logs
  - 30GB (Jan’08), 200GB (Mar’08), 15+TB (2009)
- Hive QL: SQL-like query language
Hive example

CREATE EXTERNAL TABLE records
  (last STRING, first STRING, salary INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION filename;

SELECT records.first, COUNT(1)
FROM records
GROUP BY records.first;

Q: “What is the frequency of each first name?”

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>

... ... ...
... ... ...

Monday, August 23, 2010
Hive schemas

- Data should belong to tables
  - But can also use pre-existing data
  - Data loading optional (like Pig) but encouraged

- Partitioning columns:
  - Mapped to HDFS directories
  - E.g., (date, time) → datadir/2009-03-12/18_30_00

- Data columns (the rest):
  - Stored in HDFS files

- Support for most common data types
- Support for pluggable serialization
Hive QL feature summary

- Basic SQL
  - FROM subqueries
  - JOIN (only equi-joins)
  - Multi GROUP BY
  - Multi-table insert
  - Sampling

- Extensibility
  - Pluggable MapReduce scripts
  - User Defined Functions
  - User Defined Types
  - SerDe (serializer / deserializer)
Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary
Recap
Recap

- Scalable: all
Recap

- Scalable: all
- High(-er) level: all except MR
Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
- “Schemas”: HBase, Pig, Hive, (Casc.)
  - Pluggable data types: all
Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
- “Schemas”: HBase, Pig, Hive, (Casc.)
  - Pluggable data types: all
- Easy transition: Hive, (Pig)
Related projects

Higher level—computation:
- Dryad & DryadLINQ (Microsoft) [EuroSys 2007]
- Sawzall (Google) [Sci Prog Journal 2005]

Higher level—storage:
- Bigtable [OSDI 2006] / Hypertable

Lower level:
- Kosmos Filesystem (Kosmix)
- VSN (Parascale)
- EC2 / S3 (Amazon)
- Ceph / Lustre / PanFS
- Sector / Sphere (http://sector.sf.net/)
- ...

Monday, August 23, 2010
Summary

MapReduce:
- Simplified parallel programming model

Hadoop:
- Built from ground-up for:
  - Scalability
  - Fault-tolerance
  - Clusters of commodity hardware
- Growing collection of components and extensions (HBase, Pig, Hive, etc)
Tutorial overview

- **Part 1 (Spiros): Basic concepts & tools**
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive

- **Part 2 (Jimeng): Algorithms**
  - Information retrieval
  - Graph algorithms
  - Clustering (k-means)
  - Classification (k-NN, naïve Bayes)

- **Part 3 (Rong): Applications**
  - Text processing
  - Data warehousing
  - Machine learning

NEXT: Monday, August 23, 2010